LA-UR -93-1983

Conf-930741-5

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: *ADAPTIVE TRANSFER FUNCTION NETWORKS*

AUTHOR(S): *JOHN R. GOULDING*

SUBMITTED TO: *1993 World Congress on Neural Networks*
July 15, 1993

Los Alamos

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ADAPTIVE TRANSFER FUNCTION NETWORKS

John R. Goulding
Applied Theoretical Physics, X-1 Group
Los Alamos National Laboratory, Los Alamos, NM 87545
and
Electrical Engineering Department
Portland State University, P.O. Box 751, Portland, OR 97207
goulding@eecs.ee.pdx.edu

Abstract

Real-time pattern classification and time-series forecasting applications continue to drive artificial neural network (ANN) technology. As ANNs increase in complexity, the throughput of digital computer simulations decreases. A novel ANN, the Adaptive Transfer Function Network (ATF-Net), directly addresses the issue of throughput. ATF-Nets are global mapping equations generated by the superposition of ensembles of neurodes having arbitrary continuous functions receiving encoded input data. ATF-Nets may be implemented on parallel digital computers. An example is presented which illustrates a four-fold increase in computational throughput.

Introduction

When researchers speculate about whether artificial neural networks (ANNs) will approach in complexity the human brain, something other than simulating the hundreds of billions of neurons on digital computers often enters into the conversation. Today, real-time pattern classification and time-series forecasting applications are driving ANN technology. This paper discusses a novel ANN, the Adaptive Transfer Function Network, that simulates groups of artificial neurons as arbitrary continuous functions receiving encoded input data. The process of simulating groups of neurons directly relates to the throughput of digital ANN simulations. Consequently, implementing demanding real-time applications on (parallel) digital computers is now possible.

Terminology

In this paper, the word *neurode* represents artificial models of biological neurons. Neurodes are simple threshold based signal processing devices that connect into layered *architectures* called *neural networks*. A generalized Radial Basis Function Network (RBF-Net) architecture is illustrated in Figure 1. One to P nodes make up the input layer, 1 to M neurodes make up a middle or *hidden* layer, 1 to Q summation nodes make up the output layer, and each connection (shown) is of unit value. The RBF-Net output is 1 to Q superpositions of an ensemble of radially symmetric gaussian functions. The ith component of the output vector $\underline{y}$ is defined by functions [7]:

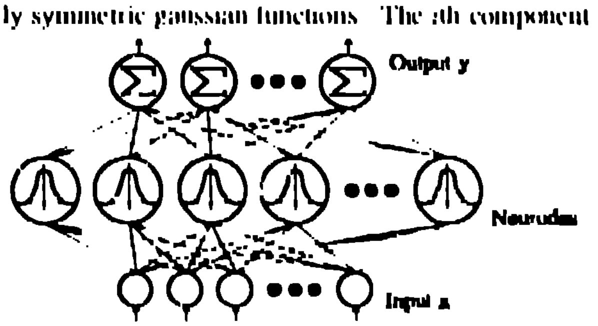

Figure 1 Radial Basis Function Network

$$y_i = \sum_{j=1}^{M} \omega_{ij}\rho_j(x) \qquad [i = 1, 2, \ldots, Q]$$

$$\rho_j(x) = \exp\left[-\alpha_j \sum_{k=1}^{P} \beta_j (x_k - x_j)^2\right]$$

RBF neurodes are comprised of four adaptive variables: ω_{ij} amplification factor, α_j activation constant, β_j width coefficient, and x_j basis function center applied to the jth neurode for every x_k component of the input vector $\underline{x}$.

In general, ensembles of neurodes build a global equation capable of mapping many discrete input to output data pairs or a continuous surface (e.g., in a *control space*). The process of adjusting neurode variables to build the global map is called *training*. Training RBF Nets entails using matrix inversion [9] or iterative back propagation of errors algorithms [6] to find the minimum global error state for a set of known input to output data pairs. To this end, RBF Nets are universal approximators [3][4][8]. That is, RBF Nets can approximate any control space to any degree of accuracy given enough neurodes. *Training sets* are generally sparsely distributed subsets of the control space. To this end, RBF Nets *generalize* untrained patterns by interpolating or extrapolating from neurode centroids

Background

For most neural network research and applications, ANNs are simulated in software on digital computers. The blackboard nature and inherent flexibility of software combined with the accessibility of digital computers has served to popularize ANN technology. Indeed, many off-the-shelf software packages exist to implement various ANN architectures. A fundamental problem exists with implementing ANNs on digital computers, however. Neural networks are parallel devices. To control real-time applications, for example, the CPU throughput must be significantly faster than the Input/Output (I/O) throughput. As the number of neurodes increases, either the CPU throughput must increase or the I/O throughput must decrease. Obviously, there is an upper limit to real-time universal approximation on digital computers. [For the purpose of the present argument, training is an off-line process, and reliability, fault-tolerance, and graceful degradation issues are not discussed.]

Many people use parallel digital computers to increase the virtual CPU throughput. RBF-Nets are easily parallelled by distributing the neurodes (as RBF-Nets) over several parallel digital computers (called *nodes*). Each node thus contributes part of the output vector $\underline{y}$. A generalized parallel RBF-Net is illustrated in Figure 2. The number of neurodes simulated (as RBF-Nets) on each of the nodes is the integer modulus of M neurodes divided by N nodes (M÷N). Typically, Node-0 connects to a host computer; so, Node-0 receives the input vector $\underline{x}$, broadcasts $\underline{x}$ to (hidden) Node-1 through Node-N simultaneously, and in a process called message routing, sequentially collects and sums the output vector $\underline{y}$. The number of nodes is a balance between the message routing time overhead of a particular parallel digital computer architecture and node throughput. As shown in Figure 2, 1 to P nodes are simulated on Node-0 to make up the input layer, 1 to M÷N neurodes are simulated (as RBF-Nets) on each of 1 to N nodes to make up a hidden layer, and 1 to Q summation nodes are simulated on Node-0 to make up the output layer. Because each node contributes a portion of the output vector $\underline{y}$, the Node-0 RBF-Net output vector $\underline{y}$ is the y_i-to-y_t superposition of an ensemble of nodes (that simulate RBF-Nets).

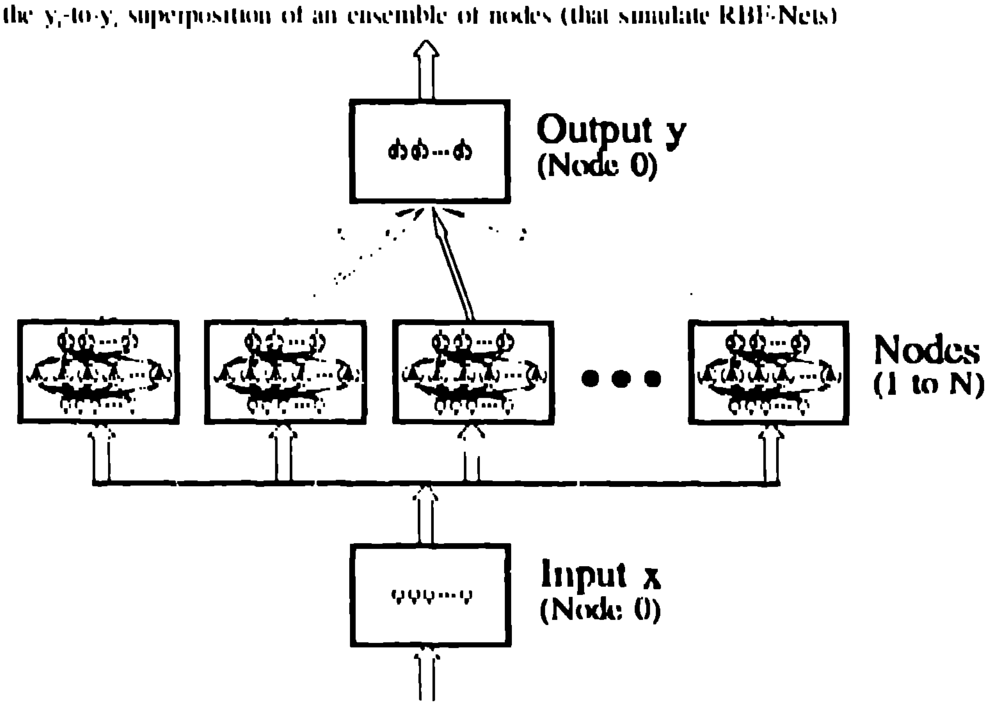

Figure 2 Parallelized Radial Basis Function Network

The tth component of the output vector $\underline{y}$ of the parallel RBF-Net is defined by functions,

$$y_t = \sum_{l=1}^{N} \zeta_{tl}(x) \qquad \zeta_{tl}(x) = \sum_{j=1}^{M \div N} \omega_{tlj}\rho_{lj}(x) \qquad \rho_{lj}(x) = \exp\left[-\alpha_{lj} \sum_{k} \beta_{lj}(x_k - x_{lj})^2\right]$$

$$[t = 1, 2, \ldots, Q]$$

To this end, there is again an upper limit to real-time universal approximation parallel digital computers albeit higher than the single digital computer.

Adaptive Transfer Functions

Consider each Node-1 through Node-N nodes of the parallel digital computer a black box. Let the neurodes be grouped so each black box contributes (all of) the ith component of the output vector $\underline{y}$. Clearly, y_i is not limited to the radially symmetric form. In fact, grouped neurodes may represent any arbitrary function [1]. For example, many gaussian transfer functions are required to generate a $y_i=\alpha_i x_k+\beta_i$ linear function. Now consider the parallelized RBF-Net as a single ANN. From the macro viewpoint, the hidden nodes may be considered (complex signal processing) neurodes with $\xi_i(\underline{x})$ arbitrary adaptive transfer functions; $\xi_i(\underline{x})$ replaces the gaussian transfer function of RBF-Net neurodes. So, this novel ANN is termed the Adaptive Transfer Function Network (ATF-Net).

It is substantially more difficult to determine an arbitrary function of P variables [$\xi_i=f(x_1,x_2,\ldots,x_k,\ldots,x_P)$] than it is an arbitrary function of 1 variable [$\xi_i=f(x_k)$]. So, let $\xi_i(\underline{x}_{Pk})$ be an arbitrary adaptive function of linear or non-linear form where $\underline{x}_{Pk}$ is the superposition of a subset of the $\underline{x}$ input vector. This approach approximates the higher dimensional transfer function with the superposition of lower dimensional transfer functions receiving encoded input data. One simple and effective data encoding method is to route a different subset of the $\underline{x}$ input vector to each ATF-neurode. This method is called *masking*, and it is commonly used in parallel digital computers. Masking is a vector multiplication process; the encoding pattern is represented as a binary set (of zeros and ones) that is multiplied with the $\underline{x}$ input vector to yield an encoded subset. Conceptually, masking is implemented in ANNs by setting the input-layer to neurode-layer connections to binary (zero or one) values. Three typical encoding masks are presented in the Table 1 for a 3-input to 3-ATF-neurode sub-architecture.

$\underline{x}_{Pk}$	Thermometer Mask	Partial Binary Mask	Partial Scatter Mask [10]
Pk = 1	001	010	011
Pk = 2	011	011	110
Pk = 3	111	100	101

Table 1 Data encoding masks.

A generalized scatter encoded [10] ATF Net is illustrated in Figure 3; however, only three input nodes are shown for clarity. One to P nodes make up the input layer, 1 to M ATF neurodes make up a hidden layer, 1 to Q summation nodes make up the output layer, and each connection (shown) is of unit value. For the purpose of conceptualization, the ATF neurodes are grouped for each y_i output to better illustrate the different scatter-encoded input received at each ATF neurode. The encoding is repeated over each grouping and is conceptualized as the input layer to ATF-neurode layer connections. [This conceptualization is in a later parallelized ATF-Net illustration.]

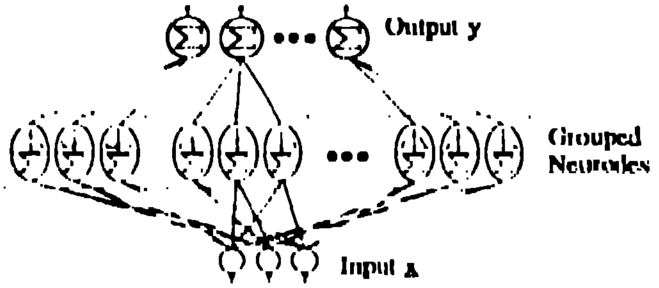

Figure 3 Adaptive Transfer Function Network

The ith component of the output vector y of the ATF Net shown in Figure 3 is defined by functions:

$$y_i = \sum_{Pk=1}^{3} \xi_{iPk}(x_{Pk}) \qquad \xi_{iPk}(x_{Pk}) = f\left(\sum_{k=1}^{P} x_k\right)$$

$$[i = 1, 2, \ldots, Q] \qquad \left| x_{Pk} \begin{cases} (0\cdot x_1,\ 1\cdot x_2,\ 1\cdot x_3) & \textit{for } Pk=1 \\ (1\cdot x_1,\ 1\cdot x_2,\ 0\cdot x_3) & \textit{for } Pk=2 \\ (1\cdot x_1,\ 0\cdot x_2,\ 1\cdot x_3) & \textit{for } Pk=3 \end{cases} \right|$$

ATF-Nets are easily parallelled by distributing the ATF-neurodes over several nodes. The total number of nodes used is determined from the number of $\underline{x}_{Pk}$ input encodings. Each node implements a single $\underline{x}_{Pk}$ encoding that is applied to every ATF-neurode of that particular node. A generalized parallel ATF-Net is illustrated in Figure 4 wherein the first three nodes show the distributed three-input scatter-encoded [10] ATF-Net of Figure 3 and the N^{th} node represents other possible encodings. One to P nodes are simulated on Node-0 to make up the input layer, 1 to P neurodes are simulated on each of 1 to N=Pk nodes to make up a hidden layer, and 1 to Q summation nodes are simulated on Node-0 to make up the output layer. The hidden nodes receive the broadcasted input vector $\underline{x}$ and implement the Pk^{th} encoding through masking. Because each node contributes a portion of the output vector $\underline{y}$, the Node-0 ATF-Net output vector $\underline{y}$ is the y_i-to-y_i superposition of an ensemble of nodes.

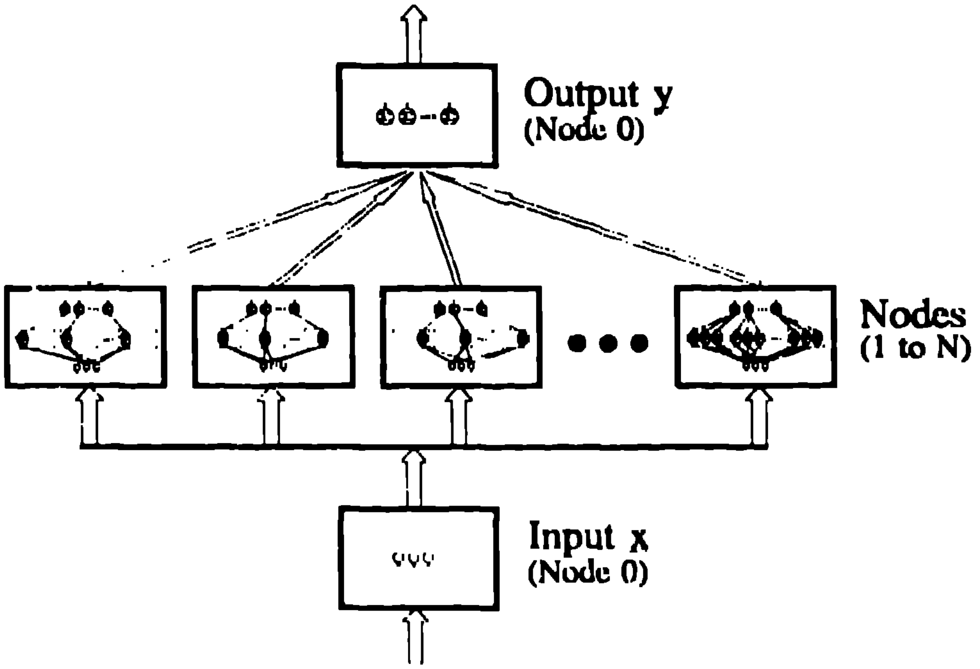

Figure 4 Paralleled Adaptive Transfer Function Network.

The ith component of the output vector $\underline{y}$ is defined for a parallel ATF-Net by functions:

$$y_i = \sum_{l=1}^{Pk} \zeta_{il}(x_{Pk}) \qquad \zeta_{il}(x_{Pk}) = f\left(\sum_{k=1}^{P} x_k\right)$$

$$\{i = 1, 2, \ldots, Q\} \qquad \left| x_{Pk} = \begin{cases} (0 \cdot x_1,\ 1 \cdot x_2,\ 1 \cdot x_1) & \mathit{for}\ Pk = l = 1 \\ (1 \cdot x_1,\ 1 \cdot x_2,\ 0 \cdot x_3) & \mathit{for}\ Pk = l = 2 \\ (1 \cdot x_1,\ 0 \cdot x_2,\ 1 \cdot x_3) & \mathit{for}\ Pk = l = 3 \end{cases} \right|$$

Training the ATF-Net

Training involves ranking an ensemble of possible linear and non-linear functions using goodness-of-fit criteria such as standard r^2, fit standard error, F statistic, and so on, to select the "best fit" $\zeta_n(\underline{x}_{Pk})$ function. Training is thus a process of applying the input and output data pairs to the ATF-Net (in one pass), distributing the values through the architecture, tabulating the x-y data pairs, curve fitting possible functions, ranking and selecting the "best fit" function, and porting the "best fit" function into the ATF-Net architecture for each of the $\zeta_n(\underline{x}_{Pk})$ functions. Once complete, a continuous and non-discrete global mapping equation results that interpolates and extrapolates between the data points not presented during training.

Comparison With Other Neurodes

The ATF-Net has been successfully implemented on both a stand-alone digital computer and a parallel digital computer using the Intel hypercube architecture. During training, the ATF-Net generates an x-y listing of data pairs for each node. A commercially available curve fitting software package is then used to determine the "best fit" function from among 3,320 built-in linear and non linear equations: 3,304 linear equations may be fit to a 50 point x-y data set in 8.3 seconds on an i80486 33MHz personal computer [5].

An interesting real time non-parallel digital computer application of the ATF-Net is learning an expert system (having crisp binary output) used to interactively design mechanical gearboxes [1][2]. Table 2 summarizes a test of non-trained data for both the conventional RBF-Nets and a scatter-encoded ATF-Net. A 4-fold increase in throughput is observed for the ATF-Net.

Property	Artificial Neural Network Type			
	Conventional RBF-Net Neurodes			ATF-Net
# Hidden Layer Neurodes/Nodes	3	12	27	7
# Connections	201	804	1809	469
Maximum Error	.667	.507	.259	.083
Average Error	.087	.031	.004	.008
Standard Deviation of Error	.169	.091	.017	.009

Table 2 Radial Basis Function and Adaptive Transfer Function Network training results.

Summary

ATF-Nets combine the inherent flexibility of digital computers, the broadcasting ability of parallel digital computers, and virtual neurode properties to simulate large ensembles of neurodes at the macro level. ATF-Nets exhibit a marked increase in overall throughput and are ideally suited for applications in real-time pattern classification and time-series forecasting. Further, the ATF-Net directly addresses the issue of learning time by training in one pass of the data.

Future demands of real-time pattern classification and time-series forecasting applications require novel approaches to ANN technology. Simulating the hundreds of billions of neurons in the human brain will perhaps require abandoning biological plausibility at the micro level. Much is gained from first studying RBF-Nets and other conventional ANNs at the micro level and then studying the properties of groups of neurodes at the macro level.

References

[1] J. R. Goulding, "Adaptive Transfer Functions," *Proceedings of International Joint Conference on Neural Networks*, IEEE Cat. No. 91CH3049-4, II-867-72, 1991.

[2] J. R. Goulding and H. Zarefar, "Development of a Mechanical Power Transmission Design Supervisor System," *Proceedings of American Society of Mechanical Engineers International Design Automation Conference*, DE Vol 23-1, 305-10, 1990.

[3] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered Neural Networks with Gaussian Hidden Units as Universal Approximations," *Neural Computation*, Vol 2, 210-15, 1990.

[4] K. M. Hornik and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol 2, 359-66, 1989.

[5] Jandel Scientific, Table Curve 3.1: Automated Curve Fitting Software, Manual No. TC3 IBR 1692, San Rafael, California, 1992, 7.

[6] R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, P. S. Lewis, and S. Qian, "Function Approximation and Time Series Prediction with Neural Networks," *Proceedings of International Joint Conference on Neural Networks*, IEEE Cat. No. 90CH2879-5, I-649, 1990.

[7] R. P. Lippman, "Pattern Classification Using Neural Networks," *IEEE Communications*, 27, 59-66, 1989.

[8] J. Park and I. W. Sandberg, "Universal Approximation Using Radial Basis Function Networks," *Neural Computation*, Vol 3, No 2, 246-57, 1991.

[9] S. Renals and R. Rohwer, "Phoneme Classification Experiments Using Radial Basis Functions," *Proceedings of International Joint Conference on Neural Networks*, IEEE, Washington, DC, I-461-7, June 1989.

[10] D. Smith and P. Stanford, "A Random Walk In Hamming Space," *Proceedings of International Joint Conference on Neural Networks*, II-465-70, 1990.